Министерство науки и высшего образования Российской Федерации  
Федеральное государственное бюджетное образовательное учреждение высшего образования

ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)

Кафедра компьютерных систем в управлении и проектировании (КСУП)

**РАЗРАБОТКА ПЛАГИНА «ЖУРНАЛЬНЫЙ СТОЛИК»**

**ДЛЯ «КОМПАС-3D»**

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

по дисциплине

«Основы разработки САПР» (ОРСАПР)

Выполнил:

студент гр. 580-3

\_\_\_\_\_\_\_\_\_\_\_ Иоч Ю.В.

«\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_ 2023 г.

Руководитель:

к.т.н., доцент каф. КСУП \_\_\_\_\_\_\_\_\_\_ Калентьев А.А. «\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_ 2023 г.

Томск, 2023

**Оглавление**

[1 Введение 3](#_Toc153572349)

[2 Постановка и анализ задачи 4](#_Toc153572350)

[3 Описание предмета проектирования 6](#_Toc153572351)

[4 Выбор инструментов и средств реализации 8](#_Toc153572352)

[5 Назначение плагина 9](#_Toc153572353)

[6 Обзор аналогов 10](#_Toc153572354)

[7 Описание реализации 12](#_Toc153572355)

[8 Описание программы для пользователя 19](#_Toc153572356)

[9 Тестирование плагина 21](#_Toc153572357)

[9.1 Функциональное тестирование 21](#_Toc153572358)

[9.2 Модульное тестирование 23](#_Toc153572359)

[9.3 Нагрузочное тестирование 24](#_Toc153572360)

[10 Заключение 31](#_Toc153572361)

[Список использованных источников 32](#_Toc153572362)

1. **Введение**

Целью данного проекта является разработка плагина для Компас-3D, который предоставит пользователям интуитивно понятный и эффективный инструмент для создания 3D-моделей «Журнального столика». Плагин должен взаимодействовать с функционалом Компас-3D, обеспечивая удобный интерфейс для конструирования и настройки отдельных деталей столика.

Плагин расширит функционал существующей системы, добавив возможность строить журнальные столики по заданным параметрам. Также это поможет автоматизировать работу производителям и проектировщикам журнальных столиков.

1. **Постановка и анализ задачи**

Задачи и поставленные сроки приведены в таблице 2.1:

Таблица 2.1 – Задачи и сроки проекта

|  |  |  |  |
| --- | --- | --- | --- |
| Задача | Наименование документа | Разработан согласно | Сроки выполнения |
| Создание технического задания | Техническое задание | ГОСТ 34.602–2020 | Не позднее 30 сентября 2023 года |
| Создание проекта системы | Проект системы | ОС ТУСУР 01-2021 | Не позднее 17 октября 2023 года |
| Модульные тесты | Программный код | RSDN Magazine #12004 | Не позднее 15 ноября 2023 года |
| Документ с тремя вариантами дополнительной функциональности плагина для согласования |
| Модульные тесты |
| Доработка плагина Создание пояснительной записки | Программный код | RSDN Magazine #12004 ОС ТУСУР 01-2021 | Не позднее 29 декабря 2023 года |
| Модульные тесты |
| Пояснительная записка |

Основным «подводным камнем» при разработке плагина стали сложности в изучении API, предоставляемого Компас-3D. В документации смешиваются API 5 и API 7, которые могут работать вместе только благодаря дублированию кода, из-за чего приходится выбирать только один. Методы и свойства также предоставляются для обеих версий API, что может ввести в заблуждение.

Положительными результатами анализа являются найденные в документации необходимые методы для реализации плагина, а также наличие некоторых их них на Github, что помогло быстрее разобраться с тем, как они работают.

Отрицательным результатом анализа послужило отсутствие разбора API и работы с ним на просторах сети Интернет. На форуме АСКОН также отсутствует необходимая информация для разработки.

1. **Описание предмета проектирования**

Журнальный столик — небольшой низкий столик, который обычно располагают перед диваном или группой кресел в гостиной, холле, для украшения интерьера и временного размещения книг, журналов, газет, еды и напитков. Согласно ГОСТу, журнальные столики используются для формирования зоны отдыха, но они также могут применяться и для приема посетителей. Иногда такой стол используют для организации небольшой столовой зоны, так как столик позволяет разместить на нем посуду и столовые приборы, хотя заниженная высота, небольшая площадь и расположение пользователей в креслах или на диванах вокруг столика ограничивают его функциональность как предмета столовой мебели [1].

Чертёж показан на рис. 3.1.

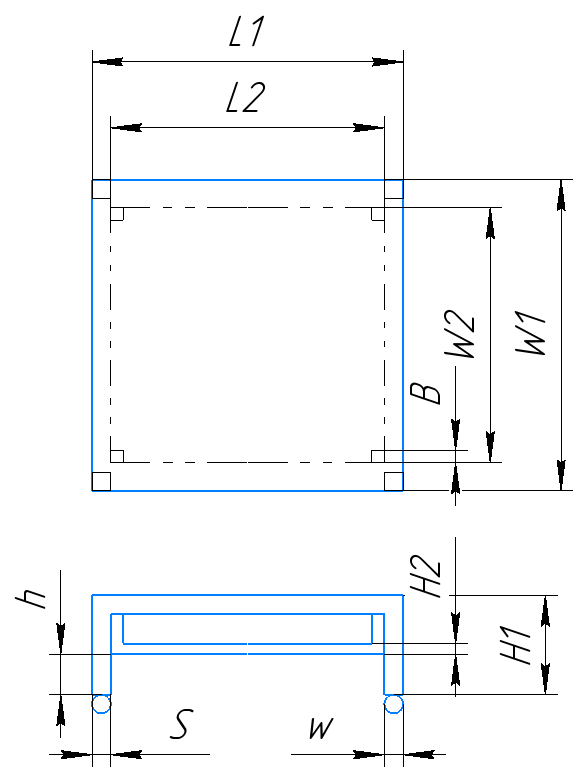


Рисунок 3.1 – Модель журнального столика

***Изменяемые параметры для*** плагина (также все обозначения показаны на рис. 2.1):

* Длина столика L1 (600 – 1200мм);
* Ширина столика W1 (600 – 1200мм);
* Высота столика H1 (400 – 500мм)
* Длина полки L2 (), длина полки не должна быть больше длины столика;
* Ширина полки W2 (), ширина полки не должна быть больше ширины столика;
* Высота полки H2 (10 – 40мм), высота полки не должна быть больше высоты столика;
* Размер ножки/поверхности столика S (30 – 50мм);
* Расстояние от пола до низа полки h (), расстояние должно быть не ниже размера ножки (равного размеру верхней поверхности) столика и не выше ширины столика за вычетом нескольких параметров;
* Размер крепления B (20 – 45 мм);
* Размер колесика w (0 – 70 мм), где 0 – отсутствие колесиков.

1. **Выбор инструментов и средств реализации**

При создании плагина использовались инструменты:

* Microsoft Word – для написания документации;
* StyleCop – для оформления кода строго по стандартам, а также улучшения читаемости кода;
* SpellChecker – для проверки орфографии в коде. Использовались русские словари;
* ReSharper – для рефакторинга и анализа кода;
* Editor Guidelines – для ограничения размера строк кода;
* Fine Code Coverage – для оценки покрытия кода модульными тестами;
* NUnit – для написания модульных тестов.

Благодаря NUnit были протестированы все классы модели.

Плагин был написан на технологии Windows Forms – это API, отвечающий за графический интерфейс пользователя и являющийся частью Microsoft .NET Framework. Приложение на Windows Forms представляет собой событийно-ориентированное приложение. В отличие от пакетных программ, большая часть времени тратится на ожидание от пользователя каких-либо действий, как, например, ввод текста в текстовое поле или клика мышкой по кнопке [2].

Плюсы технологии Windows Forms:

* Простой и понятный дизайн интерфейса;
* Набор элементов управления, достаточный для функционирования приложений;
* Событийно-ориентированная архитектура.

1. **Назначение плагина**

Назначение разрабатываемого плагина обусловлено быстрым моделированием журнального столика разных типов. Благодаря данному расширению, специалисты и мастера по журнальным столикам могут наглядно рассмотреть спроектированную модель и при необходимости перестроить под необходимые им параметры.

Таким образом, пользователи могут эффективно воплощать свои идеи, а также легко адаптировать модели столиков под конкретные требования и предпочтения.

1. **Обзор аналогов**

Косвенными аналогами плагина по построению журнальных столиков является множество написанных скриптов по автоматизированному построению мебели в системе «Базис Мебельщик» [3]. Например, Shkafscript – скрипт, по которому легко строятся столешницы (рис. 6.1). Большинство скриптов можно посмотреть на Github Bazissoft [4] (рис. 6.2). Результат работы скрипта представлен на рис. 6.3.

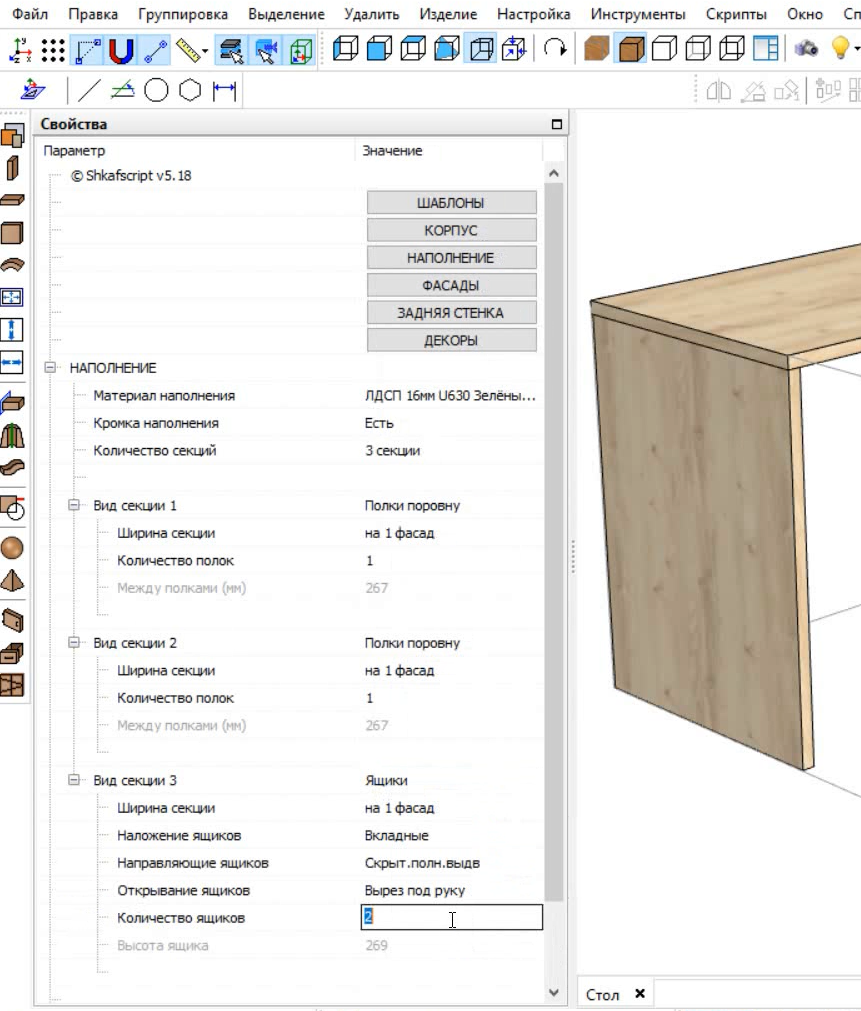


Рисунок 6.1 – Интерфейс плагина Shkafscript

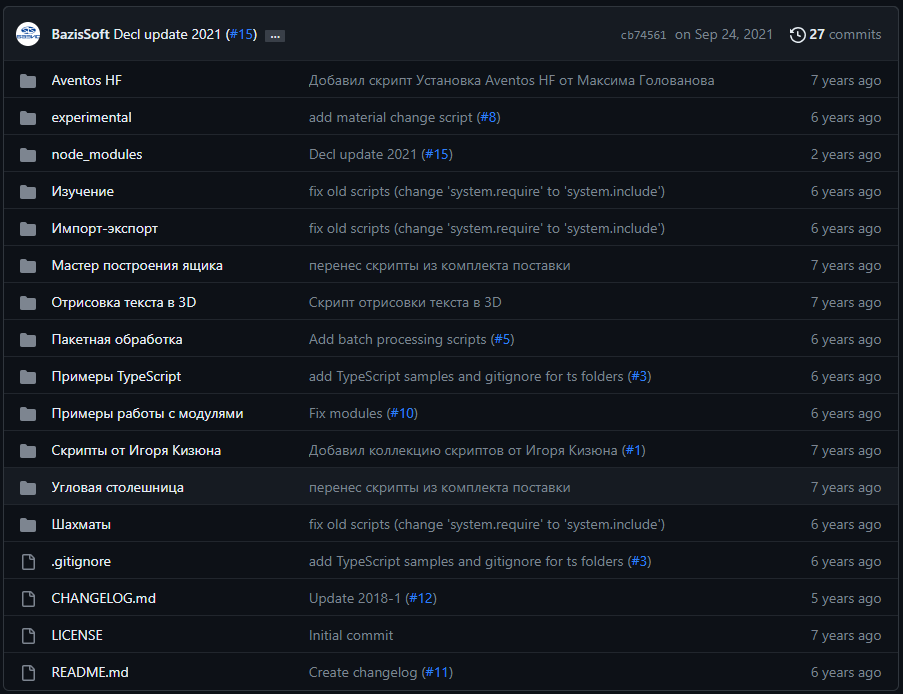


Рисунок 6.2 – Github всех плагинов для «Базис Мебельщик»

В этом плагине, как и в плагине для построения журнальных столиков, можно задавать все необходимые размеры для элементов конструкции.

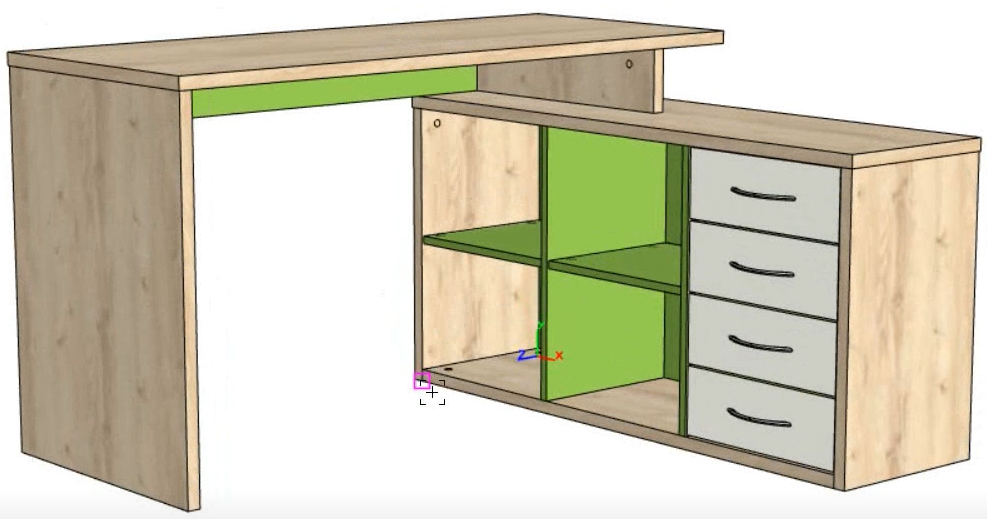


Рисунок 6.3 – Результат работы плагина

1. **Описание реализации**

После проектирования плагина получилась UML-диаграмма классов, представленная на рисунке 7.1:

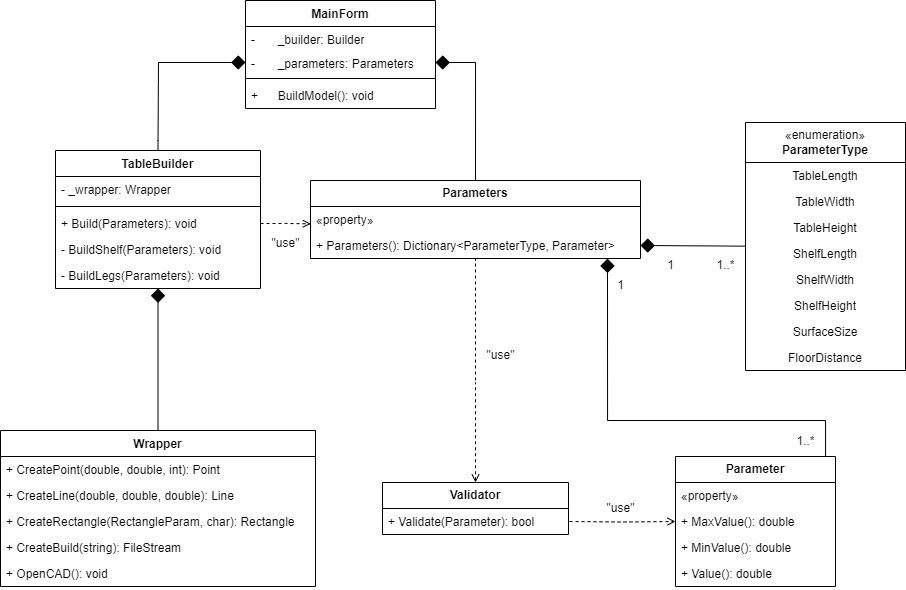


Рисунок 7.1 – Архитектура плагина после проектирования

Архитектура плагина после реализации представлена на рисунке 7.2:

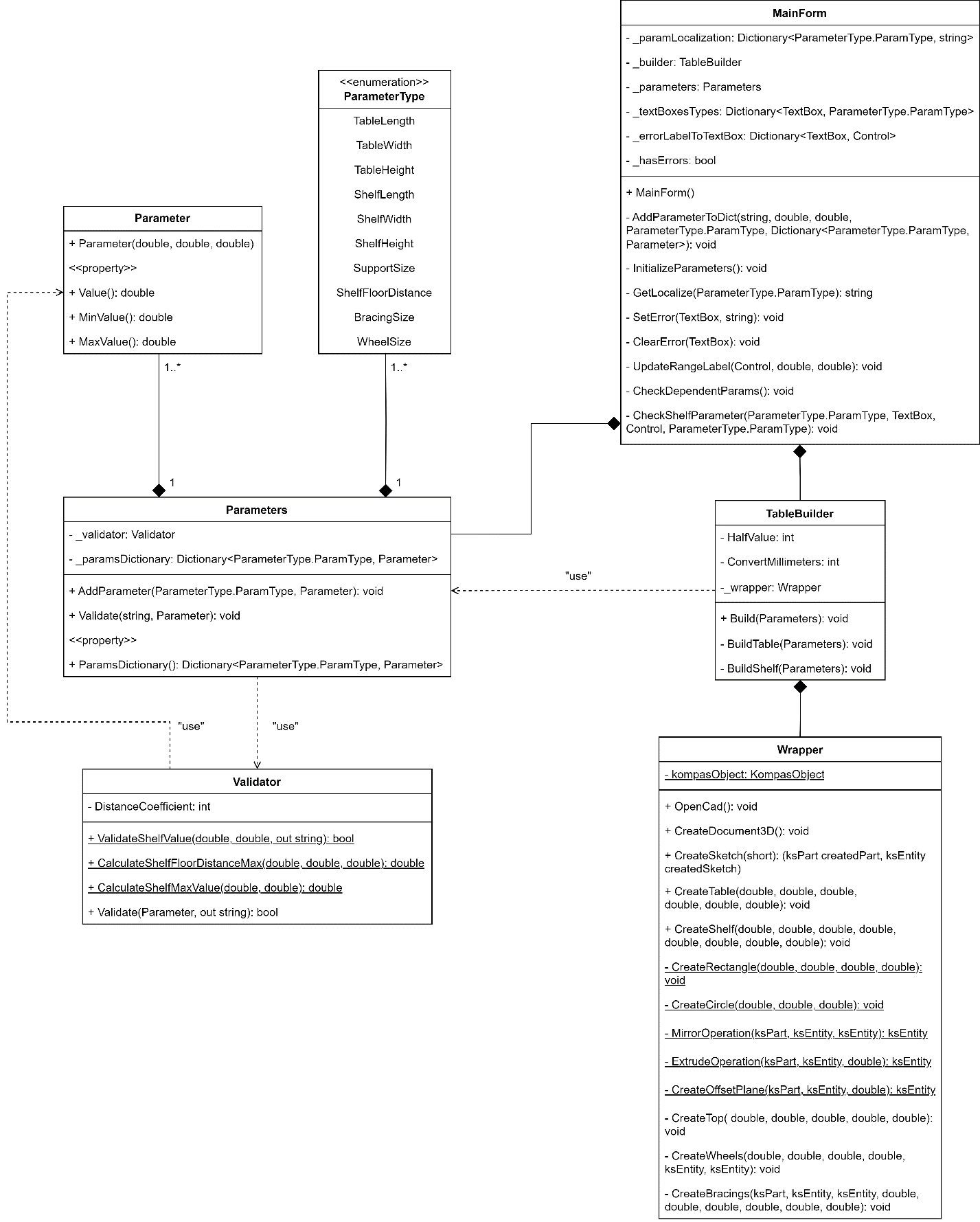
****

Рисунок 7.2 – Архитектура плагина после реализации

В процессе реализации были пересмотрены многие моменты, которые предполагались при проектировании, например, класс Wrapper’a реализует не 5, а 13 методов, каждый из которых описан в таблице 7.1; класс Validator’a валидирует и рассчитывает параметры не одним методом Validate (таблица 7.2); класс TableBuilder принимает в себя константные значения (таблица 7.3), но методы остались такими же, как предполагались при проектировании; класс MainForm включает в себя не только метод построения модели при нажатии на кнопку, но также поля (таблица 7.4) и методы (таблица 7.5); в класс Parameter был добавлен только конструктор с параметрами (таблица 7.6); класс-перечисление ParameterType включает в себя больше параметров; в класс Parameters также были добавлены методы (таблица 7.7) и поля (таблица 7.8), которых не было предусмотрено при проектировании.

Таблица 7.1 – Методы класса Wrapper после реализации

|  |  |  |  |
| --- | --- | --- | --- |
| Название | Входные параметры | Тип данных | Описание |
| OpenCad | **-** | void | Открывает Компас-3D. |
| CreateDocument3D | **-** | void | Создает 3D-документ для построения детали. |
| CreateSketch | short sketchPlane | (ksPart createdPart, ksEntity createdSketch) | Создает эскиз на плоскости. |
| CreateTable | double rectX,  double rectY,  double rectWidth,  double tableLength,  double tableWidth,  double wheelSize | void | Строит столик. |
| CreateShelf | double rectX,  double rectY,  double rectHeight,  double tableLength,  double shelfLength,  double shelfWidth,  double legSize,  double bracingSize | void | Строит полку. |
| CreateRectangle | double rectX,  double rectY,  double rectWidth,  double rectHeight | static void | Строит прямоугольник. |
| CreateCircle | double centerX,  double centerY,  double radius | static void | Строит окружность. |

Окончание таблицы 7.1

|  |  |  |  |
| --- | --- | --- | --- |
| Название | Входные параметры | Тип данных | Описание |
| MirrorOperation | ksPart part,  ksEntity entityToMirror,  ksEntity mirrorPlane | static ksEntity | Отзеркаливает объект. |
| ExtrudeOperation | ksPart part,  ksEntity sketch,  double depth | static ksEntity | Выдавливает объект. |
| CreateOffsetPlane | ksPart part, ksEntity plane, double offset | static ksEntity | Создает смещенную плоскость. |
| CreateTop | double rectX,  double rectY,  double rectWidth,  double tableLength,  double tableWidth | void | Строит столешницу. |
| CreateWheels | double rectX,  double rectWidth,  double wheelSize,  double depth,  ksEntity xozMirrorPlane,  ksEntity yozMirrorPlane | void | Строит колесики. |
| CreateBracings | ksPart part,  ksEntity plane,  ksEntity offsetPlane,  double rectX,  double rectY,  double tableLength,  double shelfLength,  double legSize,  double bracingSize | void | Строит крепления для полки. |

Таблица 7.2 – Методы класса Validator после реализации

|  |  |  |  |
| --- | --- | --- | --- |
| Название | Входные параметры | Тип данных | Описание |
| ValidateShelfValue | double shelfValue,  double shelfMaxValue,  out string error | static bool | Проверяет на корректность значения параметров, связанных с полкой. |
| CalculateShelfFloorDistanceMax | double tableHeight,  double legSize,  double shelfHeight | static double | Вычисляет максимальное значение у расстояния от пола до полки. |

Окончание таблицы 7.2

|  |  |  |  |
| --- | --- | --- | --- |
| Название | Входные параметры | Тип данных | Описание |
| CalculateShelfMaxValue | double tableSize, double legSize | static double | Вычисляет максимальное значение у параметров полки. |
| Validate | Parameter parameter, out string error | bool | Проверяет на корректность введенное значение параметра. |

Таблица 7.3 – Поля класса TableBuilder после реализации

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| HalfValue | const int | Половина значения для корректного построения. |
| ConvertMillimeters | const int | Конвертированное значение из сантиметров в миллиметры. |
| \_wrapper | Wrapper | Обертка для взаимодействия с API. |

Таблица 7.4 – Поля класса MainForm после реализации

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| \_paramLocalization | Dictionary<ParameterType.ParamType, string> | Русские названия параметров. |
| \_builder | TableBuilder | Строитель. |
| \_parameters | Parameters | Параметры модели. |
| \_textBoxesTypes | Dictionary<TextBox, ParameterType.ParamType> | Словарь, связывающий TextBox с типом параметра. |
| \_errorLabelToTextBox | Dictionary<TextBox, Control> | Словарь, связывающий TextBox с Label ошибки. |
| \_hasErrors | bool | Проверка на наличие ошибок на форме. |

Таблица 7.5 – Методы класса MainForm после реализации

|  |  |  |  |
| --- | --- | --- | --- |
| Название | Входные параметры | Тип данных | Описание |
| AddParameterToDict | string value,  double minValue,  double maxValue,  ParameterType.ParamType paramType,  Dictionary<ParameterType.ParamType, Parameter> dictionary | void | Добавляет параметр в словарь. |
| InitializeParameters | **-** | void | Инициализирует и заполняет словарь параметров значениями. |
| GetLocalize | ParameterType.ParamType type | string | Возвращает русскоязычный параметр. |
| SetError | TextBox textBox, string error | void | Устанавливает ошибку в TextBox |
| ClearError | TextBox textBox | void | Очищает TextBox с корректным значением. |
| UpdateRangeLabel | Control label,  double minValue,  double maxValue | void | Обновляет Label с диапазоном возможных значений. |
| CheckDependentParams | **-** | void | Проверяет зависимые параметры. |
| CheckShelfParameter | ParameterType.ParamType shelfParamType,  TextBox textBox,  Control rangeLabel,  ParameterType.ParamType tableParamType | void | Проверяет параметры, связанные с размерами полки и расстоянием до неё. |

Таблица 7.6 – Методы класса Parameter после реализации

|  |  |  |  |
| --- | --- | --- | --- |
| Название | Входные параметры | Тип данных | Описание |
| Parameter | double value, double minValue, double maxValue | object | Создает экземпляр параметра. |

Таблица 7.7 – Методы класса Parameters после реализации

|  |  |  |  |
| --- | --- | --- | --- |
| Название | Входные параметры | Тип данных | Описание |
| AddParameter | ParameterType.ParamType paramType, Parameter parameter | void | Добавляет параметр в словарь. |
| Validate | string paramName, Parameter parameter | void | Проверяет параметр, передавая его в Validator для вывода ошибки с конкретным параметром. |

Таблица 7.8 – Поля класса Parameters после реализации

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| \_validator | Validator | Валидатор. |
| \_paramsDictionary | Dictionary<ParameterType.ParamType, Parameter> | Словарь с параметрами. |

Все архитектурные изменения происходили по мере ознакомления с API Компас-3D и постепенным выстраиванием правильной архитектуры приложения.

1. **Описание программы для пользователя**

Для начала работы с плагином пользователю необходимо запустить его. После запуска перед ним появится интерфейс плагина (рисунок 8.1):

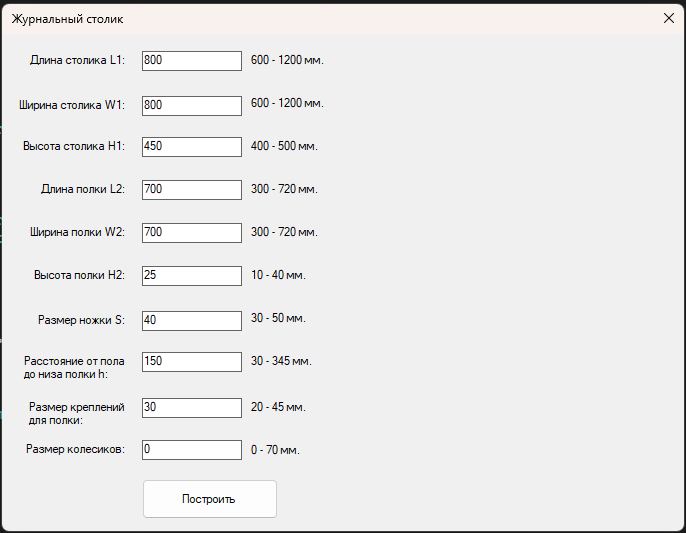


Рисунок 8.1 – Окно плагина с параметрами по умолчанию

После запуска пользователь может изменять параметры под свои предпочтения и строить необходимую модель журнального столика.

После нажатия на кнопку «Построить» открывается окно Компас-3D и начинается процесс построения. В одном окне САПР можно строить множество моделей, под каждую из которых будет создаваться отдельный документ.

При вводе некорректного значения, пользователю динамически подсвечивается возникшая ошибка (рисунок 8.2).

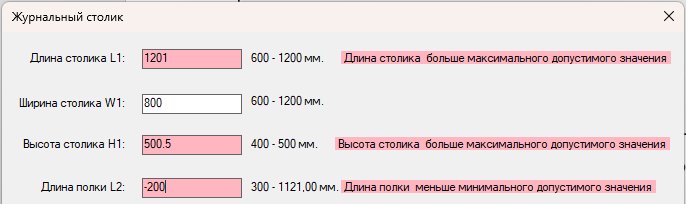


Рисунок 8.2 – Динамическая подсветка ошибок

Если при нажатии на кнопку построения САПР не запускается, возможно, проблема заключается в самой САПР – это может быть:

* Неправильная версия Компас-3D;
* Запуск Компас-3D совершается от имени администратора;
* Поврежденные файлы Компас-3D.

Модель столика при открытии Компас-3D может строиться не сразу – для решения данной проблемы стоит подождать.

1. **Тестирование плагина**
   1. **Функциональное тестирование**

Обработка ошибок плагином представлена на рисунке 9.1. Обработчик ошибок проверяет введенное значение на: вхождение значения в допустимый диапазон, ввод запрещенных символов, а также подсветку ошибок, если при вводе одного зависимого значения, другое значение стало ошибочным.

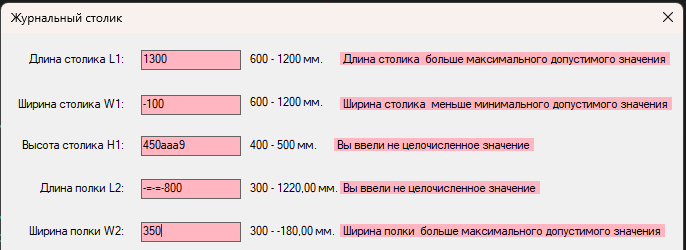


Рисунок 9.1 – Ошибки при вводе параметров

Создание модели со стандартными настройками приведено на рисунке 9.2.

Создание модели с минимальными настройками приведено на рисунке 9.3.

Создание модели с максимальными настройками приведено на рисунке 9.4.

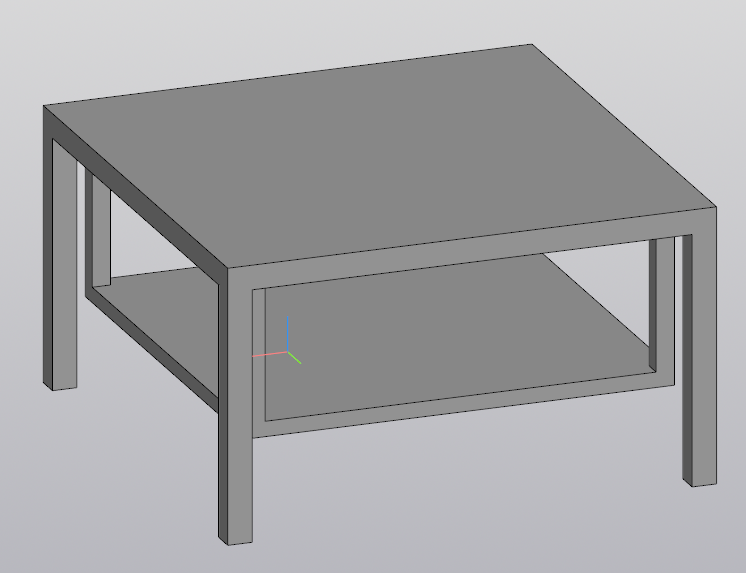


Рисунок 9.2 – Модель со стандартными настройками

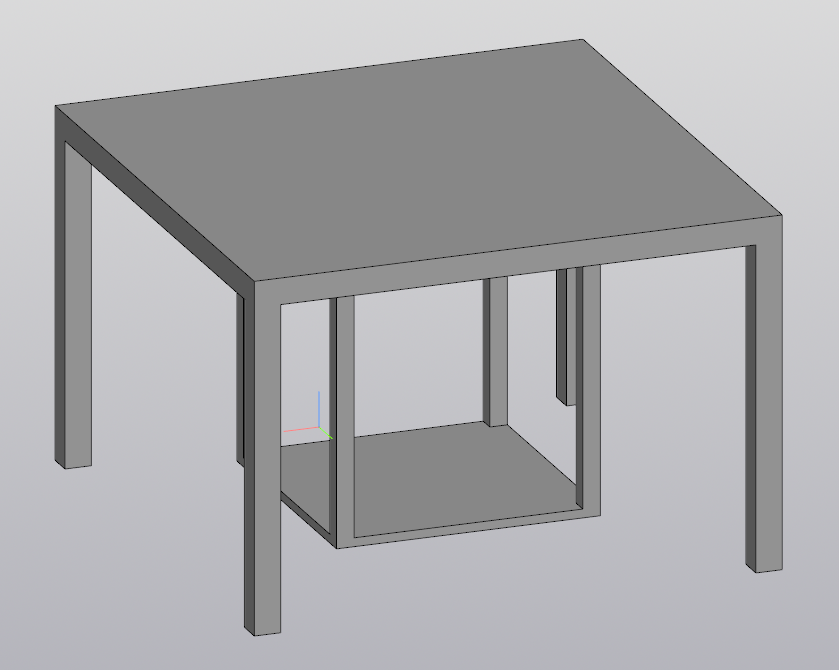


Рисунок 9.3 – Модель с минимальными настройками

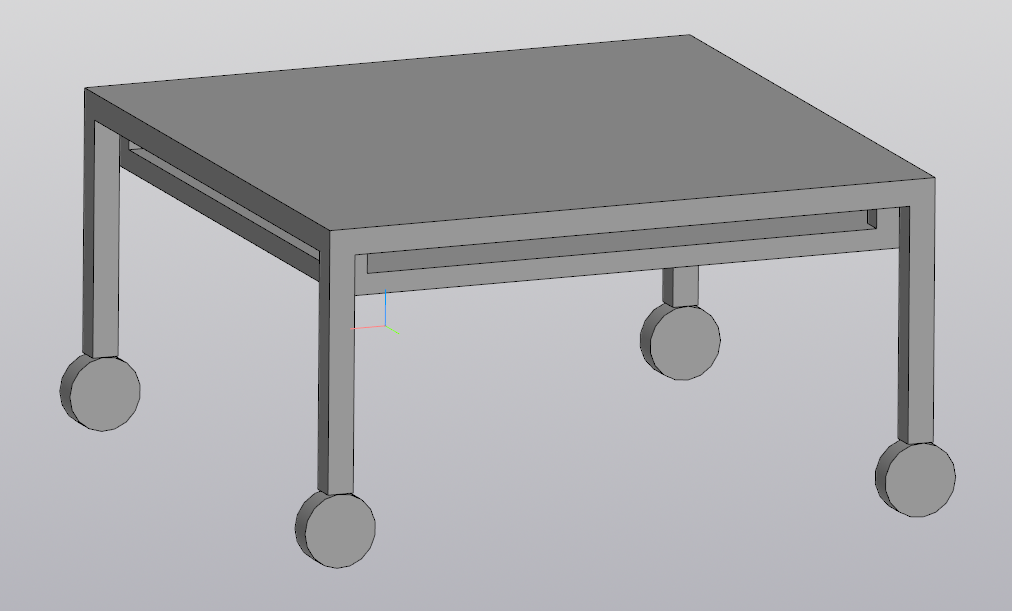


Рисунок 9.4 – Модель с максимальными настройками

* 1. **Модульное тестирование**

Итоговое количество модульных тестов для полного покрытия бизнес-логики составило 18. Количество тестов представлено на рисунке 9.5.

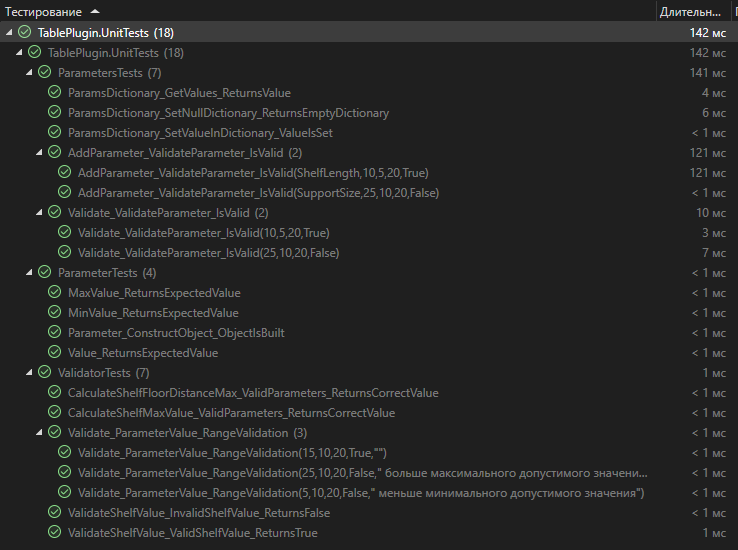


Рисунок 9.5 – Модульные тесты

В таблице 9.1 приведено описание каждого теста.

Таблица 9.1 – Описание модульных тестов

|  |  |  |
| --- | --- | --- |
| Класс бизнес-логики | Название теста | Описание теста |
| Parameter | Parameter\_ConstructObject\_ObjectIsBuilt | Проверяет, что конструктор корректно инициализирует свойства параметра. |
| Value\_ReturnsExpectedValue | Проверяет, что свойство Value возвращает ожидаемое значение. |
| MinValue\_ReturnsExpectedValue | Проверяет, что свойство MinValue возвращает ожидаемое значение. |
| MaxValue\_ReturnsExpectedValue | Проверяет, что свойство MaxValue возвращает ожидаемое значение. |

Окончание таблицы 9.1

|  |  |  |
| --- | --- | --- |
| Класс бизнес-логики | Название теста | Описание теста |
| Parameters | ParamsDictionary\_GetValues\_ReturnsValue | Проверяет геттер ParamsDictionary. |
| ParamsDictionary\_SetNullDictionary\_ReturnsEmptyDictionary | Проверяет установку значения null для ParamsDictionary. |
| ParamsDictionary\_SetValueInDictionary\_ValueIsSet | Проверяет установку непустого значения для ParamsDictionary. |
| [TestCase]AddParameter\_ValidateParameter\_IsValid | Проверяет добавление параметра в словарь. |
| [TestCase]Validate\_ValidateParameter\_IsValid | Проверяет валидацию параметра. |
| Validator | [TestCase]Validate\_ParameterValue\_RangeValidation | Проверяет валидацию значения параметра, которое меньше минимального. |
| ValidateShelfValue\_InvalidShelfValue\_ReturnsFalse | Проверяет валидацию некорректного значения параметра полки. |
| ValidateShelfValue\_ValidShelfValue\_ReturnsTrue | Проверяет валидацию корректного значения параметра полки. |
| CalculateShelfFloorDistanceMax\_ValidParameters\_ReturnsCorrectValue | Проверяет вычисление максимального значения расстояния от пола до полки. |
| CalculateShelfMaxValue\_ValidParameters\_ReturnsCorrectValue | Проверяет вычисление максимального значения параметра полки. |

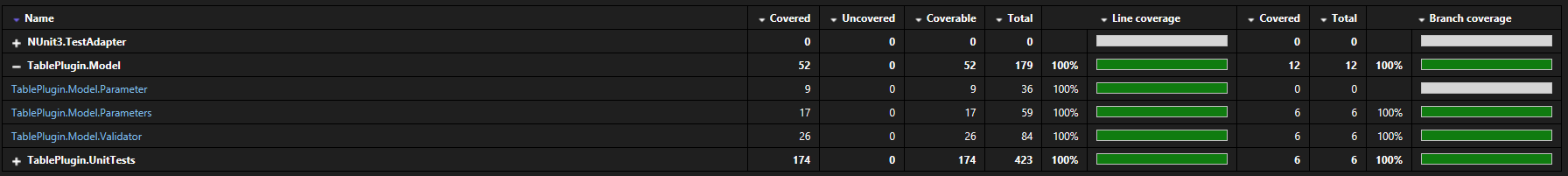
Процент покрытия кода модульными тестами представлен на рисунке 9.6. 

Рисунок 9.6 – Покрытие тестами кода

* 1. **Нагрузочное тестирование**

Для проведения нагрузочного тестирования в решение был добавлен проект StressTesting и сняты показания при различных параметрах модели: по умолчанию, минимальных и максимальных.

Тестирование проводилось на ноутбуке со следующей конфигурацией:

* Видеокарта NVIDIA GeForce 940MX;
* Процессор Intel Core i3-6100U;
* Оперативная память 8ГБ (доступно 7.65 ГБ);
* Операционная система Windows 10 Pro.

Для каждого вида параметров было выполнено приблизительно 70 построений. Большее количество построений невозможно в силу предела технических возможностей и версий комплектующих.

На рисунке 9.7 представлен график зависимости памяти ОЗУ от построения модели с настройками по умолчанию.

На рисунке 9.8 представлен график зависимости времени от построения модели с настройками по умолчанию.

Для модели с минимальными настройками графики зависимостей памяти ОЗУ от построения модели и времени от построения модели представлены на рисунке 9.9 и рисунке 9.10.

Для модели с максимальными настройками графики зависимостей памяти ОЗУ от построения модели и времени от построения модели представлены на рисунке 9.11 и рисунке 9.12.



Рисунок 9.7 – График зависимости ОЗУ (ГБ) от построения модели с настройками по умолчанию

Как видно из графика 9.7, со стандартными настройками память не понижалась первые 20 построений, после чего произошла резкая остановка обмена данными и была снижена нагрузка на оперативную память. График меняется нелинейно в связи с несовременными комплектующими. На графике все последовательности построений, идущие после 20-го построения, возрастают постепенно, что означает о использовании системой файла подкачки для распределения памяти.



Рисунок 9.8 – График зависимости времени от построения модели с настройками по умолчанию

Из графика 9.8 наблюдаются три резких скачка во времени построения моделей, что может быть связано с устаревшими комплектующими и загруженностью процессора при построении моделей со стандартными настройками. Суммарное время построения 51 модели составило 6 минут 45 секунд.



Рисунок 9.9 – График зависимости ОЗУ (ГБ) от построения модели с минимальными настройками

На графике 9.9 в конце построений можно заметить несколько последовательных остановок обмена данными для разгрузки памяти. В целом, график похож на график 9.7, но при минимальных настройках остановки обмена данными начали происходить на 3 построения раньше. Также не было замечено резких спадов памяти, как на 41-ом построении со стандартными настройками.



Рисунок 9.10 – График зависимости времени от построения модели с минимальными настройками

Из графика 9.10 можно сделать вывод, что большое количество времени при минимальных параметрах было затрачено на минимальное количество моделей, в отличие от графика 9.8, где больше всего времени было затрачено на четыре модели. На графике 9.10 самой времязатратной моделью оказалась 50-я. Общее время построения моделей с минимальными параметрами составило 3 минуты 52 секунды, что свойственно для небольших объектов. Начальный скачок во времени связан с запуском САПР.



Рисунок 9.11 – График зависимости ОЗУ (ГБ) от построения модели с максимальными настройками

Из графика 9.11 видно, что резкий спад памяти наблюдается 39-ом построении. Также можно заметить моменты, в которых использовался файл подкачки, а именно те, где график не растет и не падает. В нашем случае это наблюдается в начале до 10-го построения, и в промежутки 20-24 и 26-29 построения.



Рисунок 9.12 – График зависимости времени от построения модели с максимальными настройками

Из графика 9.12 можно сделать вывод, что на протяжении всех построений график получается более скачкообразным, чем при построениях со стандартными и минимальными параметрами. Это говорит о сложности построения модели из-за добавления колёсиков и увеличении размеров столика. Общее время построения составило 5 минут 20 секунд.

Исходя из результатов всех трех проведенных тестирований, можно сделать следующие выводы.

Во-первых, время построения модели сильно зависит от её сложности – это начинает проявляться примерно на 1/2 – 1/3 всех построений, т.к. начальные построения у всех видов построений происходят линейно.

Во-вторых, во всех трех случаях нагрузочного тестирования наблюдается достижение предела потребляемой физически доступной оперативной памяти. При минимальных и максимальных настройках до предела в 7.65 ГБ графики достигли предела ещё в самом начале, пока не произошла очистка памяти. При настройках по умолчанию это произошло позже остальных – на 20-ом построении.

В-третьих, при построении с минимальными параметрами память очищается настолько же часто, как и при построении с параметрами по умолчанию. Возможно, это связано с текущими характеристиками компьютера. При построении с максимальными настройками очищается больше памяти – пик очищения наблюдается на 39-ом построении, когда затрачиваемая память упала с 7.65 ГБ до 6.5 ГБ. После спада она продолжила линейно нарастать.

1. **Заключение**

В результате лабораторных работ был написан плагин по построению журнальных столиков в САПР Компас-3D. По занесенным пользователем параметрам происходит построение 3D-модели.

Важными результатами можно назвать получение навыков по валидации как зависимых, так и независимых параметров на форме. Правильный анализ предоставленной документации сыграл ключевую роль в успешной реализации построения 3D-моделей.

Неожиданными результатами послужили несколько версий API и выбор, на какой именно версии писать плагин. В итоге плагин был реализован на API 5, так как хоть она и не предоставляет таких методов как API 7, изучать её было гораздо проще.

Отрицательным неожиданным результатом послужило полное отсутствие информации о работе с API в интернете и на форумах.

Выполнение лабораторных работ позволило углубить знания, получить ценный опыт и развить навыки, необходимые для успешной разработки плагинов для CAD. Полученный опыт может быть применен в последующих проектах.

# **Список использованных источников**

1. Журнальный стол. Wiki [Электронный ресурс]. URL: <https://ru.wikipedia.org/wiki/Журнальный_стол>
2. Windows Forms. Wiki [Электронный ресурс] URL: <https://ru.wikipedia.org/wiki/Windows_Forms>
3. БАЗИС-Мебельщик [Электронный ресурс]. URL: <https://www.bazissoft.ru/products/bazis_mebelschik>
4. BazisSoft/Scripts [Электронный ресурс]. URL: <https://github.com/BazisSoft/Scripts.git>